# Finding a match

There are different algorithms for searching and matching elements in lists.

# Searching Sequentially

You can hopefully imagine if you were going to start writing code to do this, you might start looping from start to finish, and check each element, to see it it equals what you're looking for.

If you find a match, you'd stop looping, and return that a match was found, either with the position you found the element at, or just a boolean value, true if it was found, and false if not.

This is called a **linear search**, or **sequential**, because you're stepping through the elements, one after another.

If your elements are sorted though, using this type of linear search, is unnecessarily inefficient.

# Using intervals to Search

You split each section up, testing the values at the boundaries, and based on that, split again into smaller sections, narrowing the number of elements to test, each time.

This type of searching, in software, is called **interval searching**.

Within these two categories, sequential and interval, there are numerous existing algorithms in each.

One of the most common interval searches, is the **binary search**, which is why Java provides this search, on so many of it's collection classes.

In this search, **intervals** are continually **split into two**, hence the word binary.

# Arrays.binarySearch

The static method, binarySearch, is on the Arrays class.

We can use this method, to test if a value is already in our array, but there are some **important** things to remember.

- First, the array has to be **sorted**.
- Second, if there are duplicate values in the array, there's no guarantee which one it'll match on.
- Finally, elements must be comparable.  Trying to compare instances of different types, may lead to errors and invalid results.

# Arrays.binarySearch

This method returns:

- **The position of a match** if found.
- It returns a **-1** when **no match** was found.
- It's important to remember, that a positive number **may not be the position of the first match**.
- If your array has duplicate values, and you need to find the first element, other methods should be used.